REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	MARCH 2003	

4. TITLE AND SUBTITLE
USARIEM HEAT STRAIN MODEL : NEW ALGORITHMS INCORPORATING EFFECT OF HIGH TERRESTRIAL ALTITUDE

5. FUNDING NUMBERS

6. AUTHOR(S)
William T. Matthew, Larry G. Berglund, Ph.D., William R. Santee, Ph.D. and Richard R. Gonzalez, Ph.D.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Institute of Environmental Medicine
Kansas St, BLDG 42
NATICK, MA 01770-5007

8. PERFORMING ORGANIZATION REPORT NUMBER
T03-9

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Institute of Environmental Medicine
Kansas St, BLDG 42
NATICK, MA 01770-5007

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes modifications made to the USARIEM model to extend its applicability to high terrestrial altitude environments. Current deployments of U.S. forces to high altitude regions in and around Afghanistan provided the immediate impetus for this effort. Primary focus has been placed on elevations ranging from 0 (sea level) and 4000 meters (13,123 ft). The modifying algorithms use atmospheric pressure and are applied to the convective and evaporative heat transfer components of the USARIEM model, specifically the thermal resistance [It] and maximum evaporative power of the environment [Emax] algorithm derivations. These two modifications are minimally invasive to the USARIEM model computational engine and should be applicable to hyperbaric as well as more drastic hypobaric environments. Representation of altitude effects on convective heat transfer is accomplished by modifying the computation of total clo, [It, 1 clo = 0.157 m^2/(K·W)] to include an atmospheric pressure term that corrects for changes in air density. Measured or computed Patm, in atmospheres, is inserted in the new model as a simple multiplier for VEff to compute the altitude/pressure-sensitive total clo: ItA = Itc · (Patm· VEff)^Itvc where VEff is the effective air velocity in m/s and Itc and Itvc are the clo and clo velocity coefficients respectively for the selected uniform.

Dry thermal resistance was shown to increase with altitude, as does Emax. Thus, if all other variables are constant, dry heat loss decreases with altitude but total skin evaporation would increase. Dry skin in cool environments becomes slightly better insulated (with added clothing) and the model predicts that the wearer is protected from cold injury with increasing altitude. In hot and or sweating conditions where generally dry heat loss is small in comparison to evaporative heat loss, evaporative capacity increases, diminishing heat stress and extends the person's safe working time for the environment and activity.

14. SUBJECT TERMS
high terrestrial altitude, heat strain, model algorithms, military clothing system, C computer code

15. NUMBER OF PAGES
42

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
U	U	U	UNLIM

USARIEM Technical Report T03-9

USARIEM HEAT STRAIN MODEL: NEW ALGORITHMS INCORPORATING EFFECT OF HIGH TERRESTRIAL ALTITUDE

William T. Matthew
Larry G. Berglund
William R. Santee
Richard R. Gonzalez

Biophysics and Biomedical Modeling Division

MARCH 2003

U.S. Army Research Institute of Environmental Medicine
Natick, MA o1760-5007

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

This report describes modifications made to the USARIEM model to extend its applicability to high terrestrial altitude environments. Current deployments of U.S. forces to high altitude regions in and around Afghanistan provided the immediate impetus for this effort. Primary focus has been placed on elevations ranging from 0 (sea level) and 4000 meters (13,123 ft). The approach draws exclusively on previously published work, and is necessarily based on a combination of rational and empirically derived relationships compatible with the USARIEM model's computational structure. The modifying algorithms use atmospheric pressure and are applied to the convective and evaporative heat transfer components of the USARIEM model, specifically the thermal resistance [I_t] and maximum evaporative power of the environment [E_{max}] algorithm derivations. These two modifications are minimally invasive to the USARIEM model computational engine and should be applicable to hyperbaric as well as more drastic hypobaric environments.

The additional input parameter required for Heat Stress Monitor (HSM) instantiations of the USARIEM model is the barometric pressure measured by a new on-board sensor calibrated in atmospheres (1ATA= 760 Torr). For non-HSM instantiations, barometric pressure is computed from terrain elevation: $P_{atm} = (1 - 2.5577 \cdot 10^{-5} \cdot Z)^{5.2559}$ where P_{atm} is pressure in atmospheres and Z is terrain elevation in meters.

Representation of altitude effects on convective heat transfer is accomplished by modifying the computation of total clo, I_t (Gonzalez, 1988), to include an atmospheric pressure term that corrects for changes in air density. Measured or computed P_{atm}, in atmospheres, is inserted as a simple multiplier for V_{Eff} in the expression to compute the altitude/pressure-sensitive total clo: $I_{tA} = I_{tc} \cdot (P_{atm} \cdot V_{Eff})^{Itvc}$ where V_{Eff} is the effective air velocity in $m \cdot s^{-1}$ and I_{tc} and I_{tvc} are the clo and clo velocity coefficients respectively for the selected uniform

Representation of altitude effects on evaporative heat transfer is accomplished by modifying the computation of maximum evaporative capacity, E_{max} , to include an atmospheric pressure term. Measured or computed P_{atm}, in atmospheres, is used to produce a correction factor for the Lewis Relation (LR= 2.2 °C/Torr) embedded in the original E_{max} expression [$E_{max} = LR \cdot 6.45 \cdot A_D \cdot C_{evap} \cdot (P_{ssk} - P_{air})$]. The altitude/pressure-sensitive E_{max} is computed as: $E_{maxA} = P_{atm}^{-0.45} \cdot E_{max}$ (Nishi and Gagge, 1977).

The result is that dry thermal resistance increases with altitude, as does Emax. Thus if all other variables are constant, dry heat loss decreases with altitude but total skin evaporation would increase. Dry skin in cool environments is a little better insulated (with added clothing) and protected from cold injury with increasing altitude. In hot and or sweating conditions where generally dry heat loss is small in comparison to evaporative heat loss, evaporative capacity increases diminishing heat stress and or extends the person's safe working time for the environment and activity.

INTRODUCTION

OVERVIEW

The USARIEM Heat Strain model has been widely used to develop tailored heat stress management guidelines for the prevention of heat injury in military training and operational settings (FM 21-10, deployment manuals). Recently, the USARIEM model has been implemented in a hand-held Heat Stress Monitor (HSM) as well as the command and control oriented Operational Medicine Environmental Grid Applications (OMEGA) test bed system (5, 6). The model's principal output parameters are optimal work/rest schedules and maximum safe work time limits based on predicted body temperature changes, and hourly drinking water needed to replace predicted fluid losses from sweating. The primary inputs are environment (air temperature, humidity, wind speed, and solar load), clothing characteristics (insulation and vapor permeability), metabolic rate, and acclimatization status.

STATEMENT OF THE PROBLEM

The USARIEM model is fundamentally an empirical, operational model applied to temperature to heat stress climatic zones. It is based on the results of many years of laboratory and field studies with human volunteers over a wide range of hot environments, but at locations generally within a few hundred meters of sea level. Consequently, the wet and dry heat transfer functions represented in the model do not include consideration of the effects of high terrestrial altitude on heat strain. The military response to global terrorist activities will likely continue to involve deployments to geographic regions that lie significantly above sea level and, because such deployments may present the added heat injury risks associated with protective clothing encapsulation, there is a need to extend the USARIEM model's applicability at altitude.

OBJECTIVES

The primary objective of this report is to document modifications to the USARIEM heat strain model that are intended to extend its applicability to heat stress conditions in high terrestrial altitude environments. A secondary objective is to illustrate the results of those modifications on the model's output in a context relevant to conventional and chemical protective clothing scenarios. Scope is limited to methods and tools, and does not include a rigorous sensitivity analysis that will be the focus of a subsequent report.

APPROACH

The scientific basis for the modifications described here draws exclusively on previously published work. The approach is further constrained to a combination of rational and empirically derived mathematical relationships that are compatible with the USARIEM model's existing computational structure. The resulting modifications are focused on a practical representation of ambient

atmospheric pressure effects on the convective and evaporative heat transfer functions in the model, specifically mathematical algorithms affecting the clothing thermal insulative properties (I_t) and maximum evaporative power of the environment (E_{max}). The required additional parameter for the model's environmental input list is barometric pressure or altitude (hypobaric) effects above sea level altering the above two factors.

METHODS

DEVELOPMENT PROCEDURES AND TOOLS

Spreadsheet Version

A preliminary evaluation of the behavior of the modifications to the I_t and E_{max} functions was accomplished using a spreadsheet instantiation of the USARIEM model (Matthew et al., unpublished). This spreadsheet model version provides minute-to-minute time series outputs of intermediate calculated parameters and body core temperature as well as values for the modified I_t and E_{max} parameters during pre-defined work/rest activity scenarios. This allows a graphical view of the impact of altitude predicted core temperature responses across the relevant time domain.

C Language Version

Source Code: The C language instantiation of the USARIEM Heat Strain Model is based on transformed source code origianlly derived primarily from the MERCURY/OMEGA project's Heat Strain Decision Aid (HSDA) module (Matthew, 2000). Three source code files are compiled and linked during the build process: 1) **erf.c** that computes the error function, 2) **heatcas.c** that estimates heat casualty risk using final core temperature and the computed error function, and 3) **hsdac.c** that contains the algorithms comprising the main computational engine.

Revisions: Recent modifications to the main computational engine corrected original sweating rate computations and systematized handling of solar radiation and estimating of the Wet Bulb-Globe Temperature (WBGT) index components (Matthew et al., 2001). The current modifications to I_t and E_{max} which enable model sensitivity to altitude are implemented in the most recent version of hsdac.c, **hsdac8.c**.

Running ariem_a: The single executable file resulting from a build of hsdac8.c, erf.c, and heatcas.c is called **ariem_a.exe**. The executable reads an input file, and automatically saves the results in an output file.

Input: The input file, **ariem_a.in**, shown in Figure 1, is an eighteen line ASCII text file with the numeric input values on the left and the identifying variable, units

of measure, and comments on the left. The user edits the left hand column with a text editor such as Notepad ®, then saves that file for each run.

Figure 1. Example of the ariem_a.in file.

```
25.56       ! db (C)
90.0        ! rh (%)
1.00        ! wind speed (m/s)
25.56       ! mean radiant temp (C)
4000.0      ! Terrestrial altitude (m)
425.00      ! work rate (W)    [105, 150, 250, 425, 600]
0.00        ! work external (W)
12.00       ! acclimatization (days)     [0 - 99]
1.24        ! dehydration factor         [0, 1.24, 2.5, 4.0, 6.0]
39.0        ! max work temp (C)          [39.0 historically]
38.50       ! max work rest cycle temp (C) [38.5 historically]
36.50       ! skin temp (C) [fixed]
1.08        ! clothing parameters: clo hw_bdu=1.08
0.47        ! iclo     hw_bdu=0.47
-0.27       ! gc       hw_bdu= -0.27
0.41        ! gi       hw_bdu= 0.41
176.0       ! soldier height (cm) [152 - 218]
 70.0       ! soldier wieght (kg) [50 - 120]
```

Output Parameter Definitions: The output parameters are as follows:

MaxWk[min] = Maximum safe one-time continuous work exposure (minutes)- assumes extended recovery time in cool environment

MxWtr [qt/hr] = Hourly drinking water required to balance sweat losses during one-time max work exposure

W/R [min] = Minutes of work per hour allowed (assuming balance of the hour is spent at rest)

0 output means no sustainable work rest cycle is possible
60 output means work/rest cycles not needed

CyWtr [qt/hr] = Hourly drinking water required to balance sweat losses during work/rest cycle

0 output means work/rest cycle not applicable –(either 0 or 60 minutes output in W/R [min])

RstWtr [qt/hr] = Hourly drinking water required to balance sweat losses while at rest

RstWtrSh [qt/hr] = Hourly drinking water required to balance sweat losses while at rest in the shade

Patm = Computed local pressure in atmospheres (1ATA = 760Torr)
Heat Cas [%] = Heat casualty expectation if no action taken to implement work/rest cycle or max work time limit

Tdew [°C] = Dew point temperature

Twba[°C] = Aspirated or psychrometric wet bulb temperature

Twbn[°C] = Estimated natural wet bulb as used in computation of the Wet Bulb Globe Temperature (WBGT) index

Tg[°C] = Estimated Black Globe temperature as used in computation of the Wet Bulb Globe Temperature (WBGT) index

WBGT[°C] = 0.1 * Ta (from input) + 0.2 * Tg[°C] + 0.7*Twbn[°C]

MWkRec[min] = Estimated recovery time in minutes (in work environment) from maximum safe one-time continuous work exposure, as follows:

(1) 0 minutes output means no cooling possible: Recovery not possible in that environment
(2) >300 minutes means cooling rate too slow: Recovery not possible in that environment

MWkRecS[min] = Estimated recovery time in minutes (in work environment but in the shade) from maximum safe one-time continuous work exposure, as follows:

(1) 0 output means no cooling possible: Recovery not possible in that environment
(2) >300 minutes means cooling rate too slow: Recovery not possible in that environment

Output File: The output file, **ariem_a.out**, is a two line, tab delimited ASCII text file shown in Figure 2. The first line contains the parameter identifier text and the second line contains the output values.

Figure 2. Example of the ariem_a.out file.

MaxWk[min]	MxWtr [qt/hr]	W/R [min]	CyWtr [qt/hr]	RstWtr [qt/hr]	RstWtrSh [qt/hr]	Heat Cas [%]	.....
132	0.9	32	0.7	0.4	0.2	24.4	.....

.....	Tdew [C]	Twba[C]	Twbn[C]	Tg[C]	WBGT[C]	MWkRec[min]	MWkRecS[min]	Patm
.....	23.8	24.2	25.1	46.6	29.5	33	22	0.57

Because the output file is overwritten each time the program is run, it is necessary to save contents between runs. The horizontal format is intended to facilitate a "cut and paste" operation into a standard spreadsheet file to accumulate multiple model runs for analysis.

FUNCTION MODIFICATIONS

Terrestrial Altitude and Barometric Pressure

Altitude-related modifications to the model are based on barometric pressure. Our modifications assume local barometric pressure is not known and is approximated from known terrain elevation (1):

$$P_{atm} = (1 - 2.5577 \cdot 10^{-5} \cdot Z)^{5.2559} \quad \text{Atmospheres} \quad \text{(Eq.1)}$$

Where,

P_{atm} = pressure, atmospheres

Z = terrain elevation, meters

This addition to the source code is implemented as a defined function, PATM, in the header section of hsdac8.c. Terrestrial altitude in meters is read from the input file and P_{atm} is computed as described above. It should be noted that this function is unnecessary in HSM and OMEGA applications because automated measurements of barometric pressure would allow direct input of P_{atm} rather than Z.

Convective Heat Transfer

Representation of altitude effects on convective heat transfer is accomplished by modifying the computation of total clo, I_t, to include an atmospheric pressure term that corrects for changes in air density (1,3,8). Measured or computed P_{atm}, in atmospheres, is inserted as a simple multiplier for V_{Eff} in the expression to compute the altitude/pressure-sensitive total clo:

$I_{tA} = I_{tc} \bullet (P_{atm} \bullet V_{Eff})^{Itvc}$ clo (Eq. 2)

Where
P_{atm} = pressure, atmospheres
I_{tA} = air velocity and pressure adjusted clo for the selected uniform
V_{Eff} = effective air velocity, $m \bullet s^{-1}$
I_{tc} = clo at 1 $m \bullet s^{-1}$ for the selected uniform
I_{tvc} = air velocity modifier for clo for the selected uniform

Evaporative Heat Transfer

Representation of altitude effects on evaporative heat transfer is accomplished by modifying the computation of maximum evaporative capacity, E_{max}. Specifically, an atmospheric pressure term is added to adjust the value of the Lewis Relation (LR) embedded in the original E_{max} expression (3,8). Measured or computed P_{atm}, in atmospheres, is used to produce a correction factor for LR as previously described (1), and the resulting altitude/pressure-sensitive E_{max} is computed as:

$E_{maxA} = P_{atm}^{-0.45} \bullet LR \bullet 6.45 \bullet A_D \bullet C_{evap} \bullet (P_{ssk} - P_{air})$ Watts (Eq. 3)

where,
E_{maxA} = atmospheric pressure sensitive E_{max}, Watts
P_{atm} = pressure, atmospheres
LR = Lewis Relation, 2.2 $^{o}C \bullet Torr^{-1}$
A_D = Dubois body surface area, m^2
C_{evap} = water vapor permeability
P_{ssk} = saturation vapor pressure at T_{skin}, Torr
P_{air} = ambient air vapor pressure, Torr

RESULTS

THE SPREADSHEET IMPLEMENTATION

Body Core Temperature Profiles

Figure 3 shows spreadsheet-predicted body core temperature profiles for heat stress exposure at various terrestrial elevations wearing the battledress uniform (BDU).

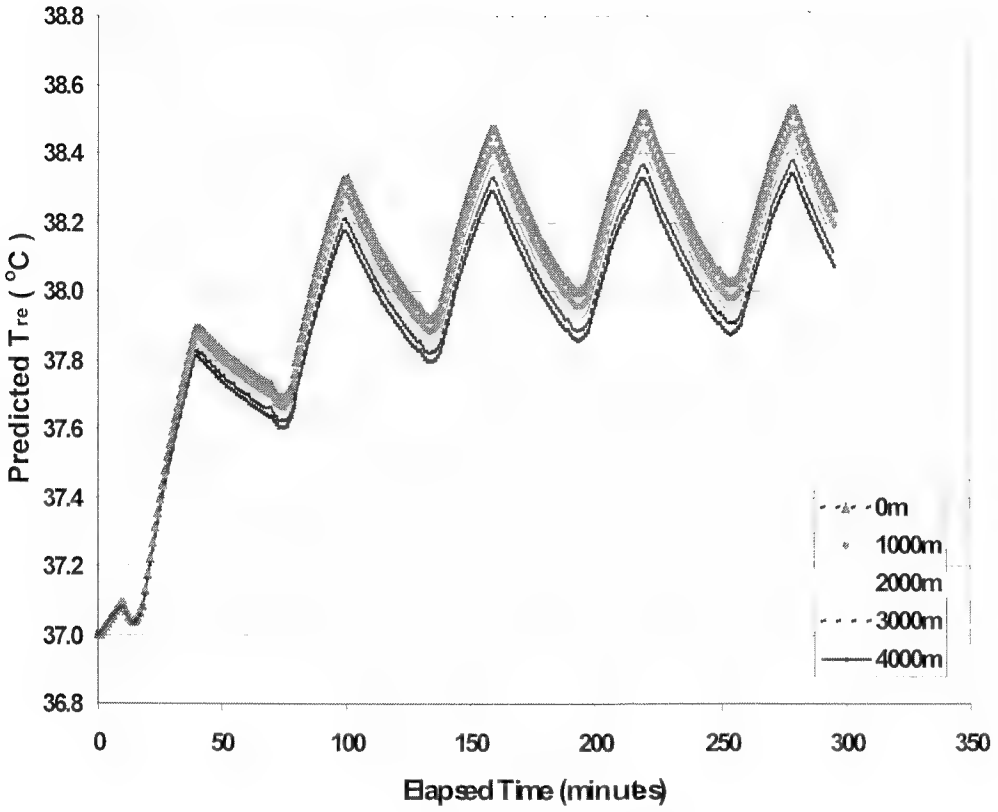


Figure 3. Spreadsheet predicted T_{re} during Work/Rest cycles in BDU at 425 Watts.

Time series input scenario begins with a ten minute rest followed by five consecutive cycles of 30 minutes of work (M= 425 Watts) and 30 minutes of rest (M= 105 Watts). Air temperature is 40 °C, RH is 20%, wind speed is 2 m/s, and mean radiant temperature is 60 °C

Figure 4 shows spreadsheet-predicted body core temperature profiles for heat stress exposure at various terrestrial elevations while wearing the new joint services light integrated suit technology (JSLIST) uniform over the hot weather BDU. Time series input scenario again begins with a ten minute rest followed by five consecutive cycles of 30 minutes of work (M= 405 Watts) and 30 minutes of rest (M= 105 Watts). Air temperature in this scene is 30 °C, RH is 40%, wind speed is 2 m/s, and mean radiant temperature is 60 °C.

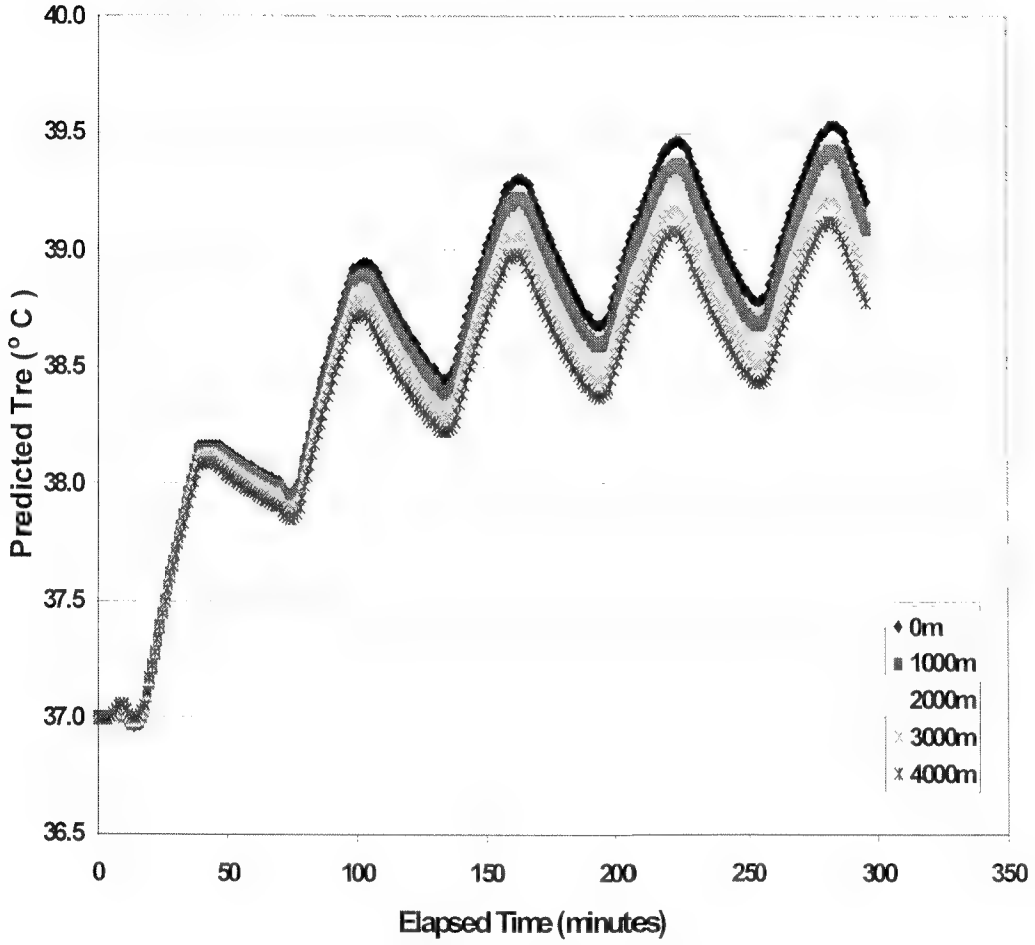


Figure 4. Spreadsheet predicted T_{re} during Work/Rest cycles in JSLIST over BDU at 425 Watts.

Modulated by time delay algorithms which change whenever there is a transition to a different value for metabolic rate, the core temperature profile is fundamentally driven by the model's internally predicted final equilibrium rectal temperature parameter, $T_{re,f}$. It represents the theoretical final steady state rectal temperature for a static condition set. For the BDU scenario the sea level and 4000 meter $T_{re,f}$ values were 37.55 and 37.46 °C respectively at 105 Watts and 39.07 and 38.82 °C at 425 Watts. For the JSLIST scenario the sea level and 4000 meter $T_{re,f}$ values were 37.75 and 37.64 °C respectively at 105 Watts and 40.40 and 39.92 °C at 425 Watts. The model output clearly suggests that the net effect of the high terrestrial altitude is to improve heat transfer and mitigate heat strain in both conventional (BDU) and protective encapsulation (JSLIST) scenarios.

Water Requirements

As in the C language version of the ARIEM operational model, prediction of water requirements in the spreadsheet version has no time dependency. It is based on prediction of an average equilibrium sweat rate (Shapiro, 1982) that remains constant for a particular environment, clothing type, and work load. Response lag time algorithms have not been implemented and, consequently, transitions are abrupt when viewed in time series format. Figure 5 shows the spreadsheet-predicted hourly drinking water requirements for the working (M= 425 Watts) and resting (M=105 Watts) conditions in BDU. Air temperature is 40 °C, RH is 20%, wind speed is 2 $m \cdot s^{-1}$, and mean radiant temperature is 60 °C.

Figure 5. Spreadsheet-predicted water requirements in BDU at 425 Watts and at rest.

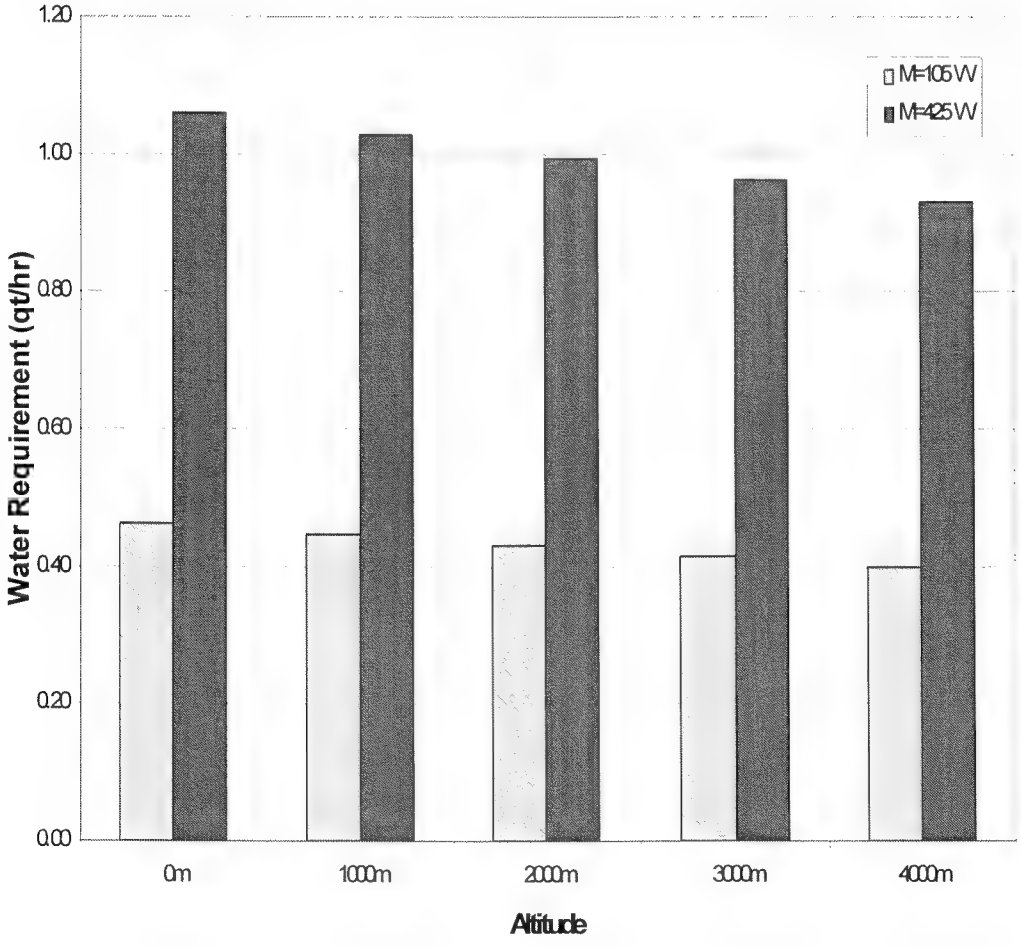
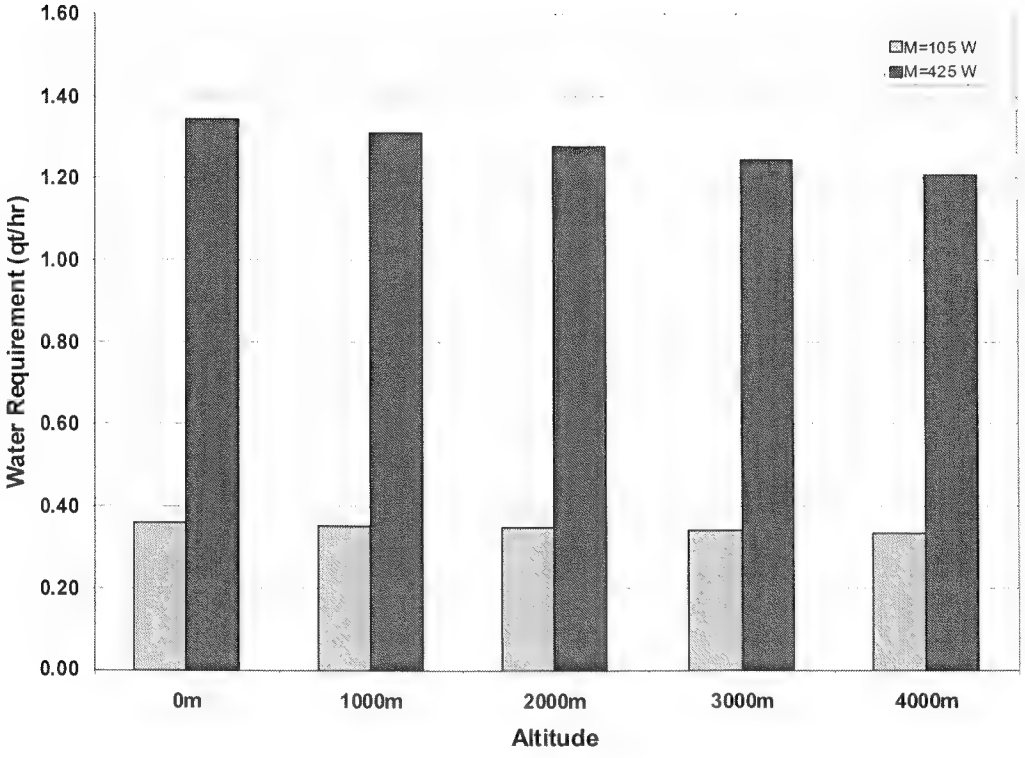


Figure 6 shows the spreadsheet- predicted hourly drinking water requirements for the working (M= 425 Watts) and resting (M=105 Watts) conditions in JSLIST worn over BDU. Air temperature is 30 °C, RH is 40%, wind speed is 2 m/s, and mean radiant temperature is 60 °C.

Figure 6. Spreadsheet-predicted water requirements in JSLIST over BDU at 425 Watts and at rest.



Differences between the sea level and 4000 meter water requirements are quite small for both the conventional (BDU) and protective encapsulation (JSLIST) scenarios.

THE C LANGUAGE IMPLEMENTATION

Source Code

Source code listings for this updated C language version of the USARIEM heat strain model are provided at Appendix A. The source code listing for hsdac8.c contains several blocks of 'dead code' that have been retained to help document revision history. While the core functionality of the code is adequate for the simplest analytical tasks, it is clear that a rigorous, disciplined re-write of the computational engine is in order. The addition of a graphical user interface

(GUI) and input/output facilities to automate multiple 'case' runs would enhance the overall usefulness of the model. Future modifications are planned to implement this aspect of the model.

Output Products

The executable C language version of the USARIEM heat strain model (ariem_a, compile date 11/30/2001) provides tabular output for a single input situation or 'case'. Output for each 'case' includes maximum safe work time, optimal work/rest cycle limit, associated water requirements as well as a heat casualty probability risk assessment, and a computed estimate of the ambient wet bulb-globe temperature (WBGT) index. Table 1 depicts output results for a clothing configuration consisting of the JSLIST uniform worn over the BDU. Each line in the table used the ariem_a executable with a different input file and contents of the resulting output file were 'pasted' into a summary spreadsheet file. That process was repeated for each of the five terrestrial altitudes within each of the three assumed metabolic rates

Table 1. The ariem_a model output sensitivity to high terrestrial altitude in chemical protective posture at three different work rates.

JSLIST over BDU at Ta= 30 °C, RH=40%, Va=2 m/s, Tmr=60 °C

M=250 Watts

Altitude (m)	MxWk[min]	MxWtr [qt/hr]	W/R [min]	CyWtr [qt/hr]	Heat Cas [%]
0	300	0.8	60	0.8	7.7
1000	300	0.8	60	0.8	6.4
2000	300	0.8	60	0.8	5.2
3000	300	0.8	60	0.8	4.2
4000	300	0.7	60	0.7	3.3

M=350 Watts

Altitude (m)	MxWk[min]	MxWtr [qt/hr]	W/R [min]	CyWtr [qt/hr]	Heat Cas [%]
0	81	1.1	24	0.7	59.0
1000	85	1.1	25	0.7	52.8
2000	91	1.1	27	0.7	46.4
3000	98	1.0	28	0.7	39.9
4000	108	1.0	31	0.7	33.5

M=425 Watts

Altitude (m)	MxWk[min]	MxWtr [qt/hr]	W/R [min]	CyWtr [qt/hr]	Heat Cas [%]
0	58	1.3	19	0.7	96.1
1000	60	1.3	19	0.7	93.9
2000	62	1.3	20	0.7	90.6
3000	64	1.2	21	0.7	86.1
4000	68	1.2	23	0.7	80.2

DISCUSSION

VERIFICATION AND VALIDATION

Verification issues for the USARIEM heat strain model are somewhat complicated by its empirically-derived, population-based, development history. In order to provide a margin of safety for those individuals whose response would be greater than the statistical average response, the model was intentionally 'tuned' to over-predict the final equilibrium T_{ref} by as much as two standard deviations. The resulting optimal work/rest cycles and maximum safe work time guidance are therefore very conservative. While this is a desirable and probably necessary characteristic for a tactical decision aid application, it results in a somewhat disconcerting level of error when averaged field or laboratory response measurements are compared with model output. Previous work has indicated that an adjustment to the denominator of the K_{work} expression from 120 to 225 is sufficient to bring the time dependent outputs into line with averaged response measurements (Gonzalez, 1997; Cadarette, 1999). The ability to switch between the conservative and statistical average response formulations for K_{work} depending on application context (i.e. either operational or research/validation) would extend the global utility of the model. Nevertheless, a systematic revision of the model to accurately predict mean T_{re} response coupled with a statistically valid stochastic representation of core temperature probabilities in the high, or "risk tail" portion of the response distribution would probably result in a more robust risk assessment paradigm. For the sake of consistency with a large number of recent modeling analysis tasks, the current instantiation employs the original, conservative 120 value in the K_{work} expression.

USER APPLICATIONS

The range of thermal strain applications for the USARIEM model is considerably expanded with the addition of the algorithms to instantiate sensitivity to high terrestrial altitude following basic heat transfer principles outlined by previous work (Nishi and Gagge, 1977; Gonzalez, 1988). While these algorithms do not consider non-thermal health risks associated with living or working at high terrestrial altitudes, they do extend the heat strain model working domain to terrestrial elevations that are consistent with emerging geo-political deployment scenarios. The additional improvements and additions to the current ARIEM heat stress model is ideally applicable to various Objective Force Warrior requirements and Land Warrior clothing systems planned for the future. The implementation of the model in C language options is also a marked improvement from the previous versions.

CONCLUSIONS

The simple ASCII text input/output file interface to the ariem_a model severely limits its utility for anything more than the simplest analytical tasks. There is a need to design and implement a Graphical User Interface (GUI) and extensions that allow a batch mode capability that will automate preparation of a single, larger input file consisting of user-specified multiple domain ranges and step intervals

REFERENCES

1. ASHRAE. (ASHRAE 2001, Handbook of Fundamentals, SI edition, p6.2, eq 2)

2. Cadarette, B.S., S.J. Montain, M.A. Kolka, L. Stroschein, W. Matthew, and M.N. Sawka. Cross Validation of USARIEM Heat Strain Prediction Models. *Aviation, Space and Environmental Medicine*, 70:996-1006, 1999.

3. Gonzalez, R.R., Biophysics of Heat Transfer and Clothing Considerations, In Human Performance Physiology and Medicine at Terrestrial Extremes, Pandolf,Sawka, Gonzalez,Eds. Benchmark Press, 1988, pp76,77)

4. Gonzalez, R.R., T.M. McClellan, W.R. Withey, S.K. Chang, and K.B. Pandolf. Heat strain models appropriate for protective clothing systems: comparison of core temperature response. *J. Appl. Physiol, 83(3): 1017-1032, 1997.*

5 .Matthew, W.T., J.A. Gonzalez, R.R. Gonzalez, G. Bates and C. Gazey. *Field Testing of a Prototype Heat Stress Monitor: System Performance and Applicability to Commercial Mining in Australia*. Natick, MA: USARIEM Technical Report T99-7, June 1999.

6. Matthew, W.T. *MERCURY User's Guide- Version V1.11alpha*. Natick, MA: USARIEM Technical Note TN007, June, 2000.

7. Matthew, W.T., W.R. Santee, and L.G. Berglund. Solar Load Inputs for USARIEM Thermal Strain Models and the Solar Radiation-Sensitive Components of the WBGT Index. Natick, MA: USARIEM Technical Report T01-13, June 2001.

8. Nishi, Y., and A.P. Gagge. Effective temperature scale useful for hypo- and hyperbaric environments. Aviat. Space Environ. Med. 48(2): 97-107, 1977.

9.. Shapiro, Y., K.B. Pandolf, and R. F. Goldman. Predicting sweat loss response to exercise, environment, and clothing. *Eur. J. Appl. Physiol, 48:83-96, 1982.*

APPENDIX A

heatcas.c

```
double erf(double XX);


float heatcas(float Tmxcore)

/*************************************************************************
*****
  computes heat strain casualty

*************************************************************************
****/
{
static float Tmean = 39.5F, sigma = 0.5198F, sqrt2 = 1.4142135F;
float heat_cas, arg;

      arg = (Tmxcore - Tmean) / (sqrt2 * sigma);
    heat_cas = (float)erf(arg) / 2.0F + 0.5F;
      heat_cas *= 100.0F;                  /* => % */

      return( heat_cas );
}
```

Erf.c

```
#include <math.h>

double erf(double XX)
/*************************************************************************
  C Computes the value of the error function at the point XX
  Ref: C. Hastings, Approximations for Digital Computers,
       Princeton Univ. Press, Princeton, NJ, 1955
  Referenced by: DSDIST, XYPVPR
  External References: None

*************************************************************************/
{
double ERFCTN, SIGN, X, X2, X3, X4, X5, X6, Y;
static double A1=0.0705230784, A2=0.0422820123, A3=0.0092705272,
              A4=0.0001520143, A5=0.0002765672, A6=0.0000430638,
              PI=3.14159265, XMIN=1.E-5, XMAX=5.;

    X=XX;
    if (X == 0.)
      return(0.0);

    SIGN = X / fabs(X);
    X = fabs(X);

    if (X < XMIN)
      ERFCTN = 2. * X / sqrt(PI);
    else if (X > XMAX)
```

```
      ERFCTN=1.0;
    else {
      X2=X*X;
      X3=X2*X;
      X4=X3*X;
      X5=X4*X;
      X6=X5*X;

      Y=1.+A1*X;
      Y=Y+A2*X2;
      Y=Y+A3*X3;
      Y=Y+A4*X4;
      Y=Y+A5*X5;
      Y=Y+A6*X6;

      Y = pow(Y, 16.0);
      Y = 1. / Y;

      ERFCTN = 1. - Y;
        }

    X = SIGN * X;
    ERFCTN = SIGN * ERFCTN;

      return( ERFCTN );
}
```

hsdac_8.c

```
/*************************************************************************
program: ariem_a;  This file is HSDAC8.C :Computational Engine Heat
Strain Decision Aid C, version: 8
 created: October 2001

History,USARIEM model source code:
BASIC Language engine- L.A. Stroschein, USARIEM:
C Language PC version of firmware implementation for Heat Stress Monitor
(HSM) - K. Honeyager, Under Contract, SwRI
C Language functional extensions for MERCURY/OMEGA - J.R. Furlong, Under
Contract, SAIC
C Language engine modifications for high terrestrial altitude - W.
Matthew & L. Berglund, USARIEM

 REFERENCES

1) Pandolf, K.B., L.A. Strochein, L.L. Drolet, R.R. Gonzalez, M.N.
Sawka, "Predictive Modeling of Physiological Responses and Human
Performance in the Heat," Comp. Biol. & Med., 6:319-329, 1986.

*************************************************************************/

#include    <stdio.h>
#include    <string.h>
#include    <math.h>

#define data
```

```
#define code
#define xdata
#define idata

#define fin "ARIEM_a.in"
#define fout "ARIEM_a.out"
#define LBUF 128
#define DUBOIS(ht,wt) 0.007184F * (float)(pow(ht, 0.725) * pow(wt,
0.425))
#define PATM(alt_m) (float)pow((1.0 -0.000025577 * alt_m),5.2559)

#define LEWIS_COR(patm) (float)(pow(patm, -0.45))

#define SUCCESS 1
#define FAILURE 0
#define STEFAN_BOLTZMANN 5.67e-8

float heatcas(float Tmxcore);

float tmtry;

float Ereqs;

float Emaxs;


typedef struct inputs {
      float Ta;
      float rh;
      float ws;
      float mrt;

      float alt_m;
      float workrate;
      float workextn;
      float acclim;
      float dehyd;
      float Tmxwork, Tmxcyc, Tmxmetrate, Tskin;
      float clo, iclo, gc, gi;
      float ht, wt;
    } INPUTS;

INPUTS hsm_in;
FILE *stream;
char buffer[LBUF+1];

typedef struct  algo_variables {
      float TrefWK;
      float TrefRY;
      float TrefAWK;
      float TrefARY;
      float WtrWK;
      float WtrRY;

      float WtrRYSh;
      float CP;
```

```
      float CPSh;
      float TDWK;
      float TreoCWK;
      float KWK;
      float DTreWK;
      float TDRY;
      float KRY;
      float DTreRY;
      float TmCWK;
      float TmCRY;

      float TreoCRY;
   float Tskin;
} HSMALGO_STRUCT;

/*
 * Function prototypes for Heat Stress Monitor functions.
 */
void CalcBaseValues (HSMALGO_STRUCT *);
void DoTheRest (HSMALGO_STRUCT *);
void ComputeAspiratedWetBulb (float *);
void GetSkinTemp (HSMALGO_STRUCT *);
void ComputeDewPoint();
int  ComputeNaturalWetBulb();
int  HSPrediction(void);
int  ComputeGlobe (void);
float SVP (float T);
float Veff (float);
float It(float,float);
float Cevap (float);
float U (float);
float Hrc (float, HSMALGO_STRUCT *, float Asold);
float Ereq (float, float, float, float);
float Emax (float patm, float, float, HSMALGO_STRUCT *, float Asold);

float Tref (float, float, float, float, float, float);
float DTref (float, float);
float WTR (float, float, float Asold);
float MxWK (float, float, float, float, float);
float MxWRKRY (float, float, float,float);

float TreRY (float t, float TreoCRY, float DTreRY, float KRY, float
TDRY);
float TreWK (float t, float TreoCWK, float DTreWK, float KWK, float
TDWK);
float round (float, unsigned char);
float f_wetbulb(float hconv, float hevap, float Ta, float Trk4, float
Pdp,
                        float Twb);

/*
 * Structure definitions.
 */
typedef struct clothing_struct {
      char  *text;
      float itc;
      float itvc;
```

```
      float imc;
      float imvc;
} clothing_table;

typedef struct clothing_types {
      char  *text;
      clothing_table *clo_lst;
} clothing;

/*typedef struct work_struct {
      char  *text;
      float metabolic_rate;
} work_table;*/

struct select {
      unsigned char clothing_type_idx;
      unsigned char clothing_idx;
      /*unsigned char work_rate_idx;*/
};

struct hsm_data {
      float globe;
      float wbgt;
      float wet_bulb;
      float awet_bulb;

      float dry_bulb;
      float wind_speed;
      float relhum;
      float mrt;

      float alt_m;
      float dewpoint;

      float Lcor;
};


/*
 * Define the MISCELLANEOUS clothing option list.
 */
clothing_table code misc_list[1] = {
      "MERCURY INPUT      ",  -99.0, -99.0, -99.0, -99.0,
      };

clothing code clothing_list[1] = {
      "MISCELLANEOUS      ", misc_list,
      };

/*work_table code work_list[] = {
      "LEVEL 1            ", 105.0,
      "LEVEL 2            ", 150.0,
      "LEVEL 3            ", 250.0,
      "LEVEL 4            ", 425.0,
      "LEVEL 5            ", 600.0,
      NULL,              0.0*/
/*};*/
```

```
/*
 * Structure containing menu selections used in the calculation of
 * work/rest and water requirements.
 */
xdata struct select input_selections;
struct hsm_data measurements;

/* global variables */
xdata float MRT;

xdata float patm;
/* Heat stress prediction results */
xdata int work_rest;
xdata float water_req;
xdata int max_work;
xdata float max_water;

xdata float rest_water;

xdata float rest_water_sh;

xdata float maxwk_rcy;

xdata float maxwk_rcy_sh;
xdata float heat_cas;


main (int argc, char **argv)

{
/*int argi; */
/*unsigned char string[80]; */
clothing_table *cl_table;


      /* read ariem_a.in */
    stream = fopen(fin, "r");
    if (stream == NULL) {
        strcpy(buffer, fin);
      printf("Can not open %s\n", buffer);
        exit(1);
      }

    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.Ta);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.rh);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.ws);
      fgets(buffer, LBUF, stream);

      sscanf(buffer, "%f", &hsm_in.mrt);

      fgets(buffer, LBUF, stream);
```

```
    sscanf(buffer, "%f", &hsm_in.alt_m);

    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.workrate);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.workextn);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.acclim);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.dehyd);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.Tmxwork);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.Tmxcyc);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.Tmxmetrate);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.Tskin);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.clo);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.iclo);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.gc);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.gi);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.ht);
    fgets(buffer, LBUF, stream);
    sscanf(buffer, "%f", &hsm_in.wt);

    fclose(stream);

      measurements.globe = 42.8896F;
      measurements.dry_bulb = hsm_in.Ta;
      measurements.relhum = hsm_in.rh;
      measurements.wind_speed = hsm_in.ws;
      measurements.mrt = hsm_in.mrt;                    /* [C] */
      MRT = hsm_in.mrt;                                 /* global MRT */

      patm = PATM(hsm_in.alt_m);

      measurements.Lcor = LEWIS_COR(patm);

      /* I've arbitrarily chosen to use the miscellanceous category
         to store clothing input - jrf */
      /* All lists but miscellaneous deleted for v0.61 */
    input_selections.clothing_type_idx = 0;     /* miscellaneous list */
    input_selections.clothing_idx = 0;          /* MERCURY INPUT */

      /* select the table */
    cl_table =
clothing_list[input_selections.clothing_type_idx].clo_lst;

      /* load the table w/ clothing coefficients */
    cl_table[input_selections.clothing_idx].itc = hsm_in.clo;
```

```
      /* clothing velocity coefficient */
    cl_table[input_selections.clothing_idx].itvc = hsm_in.gc;

      /* clothing permeability coefficient */
    cl_table[input_selections.clothing_idx].imc = hsm_in.iclo;

      /* clothing permeability velocity coefficient */
    cl_table[input_selections.clothing_idx].imvc = hsm_in.gi;

      /* set the work rate index
      if (hsm_in.workrate < 150.0F)              resting
      input_selections.work_rate_idx = 0;
      else if (hsm_in.workrate < 250.0F)         v. light
      input_selections.work_rate_idx = 1;
      else if (hsm_in.workrate < 425.0F)        light
      input_selections.work_rate_idx = 2;
      else if (hsm_in.workrate < 600.0F)        moderate
      input_selections.work_rate_idx = 3;
      else
      input_selections.work_rate_idx = 4;  heavy */

    if (HSPrediction() == FAILURE)  /* indicate FAILURE to MERCURY */
        max_work = -999;

      /* write hsdac.out */
    stream = fopen(fout, "w");
    if (stream == NULL) {
        strcpy(buffer, fout);
      printf("Can not open %s\n", buffer);
        exit(1);
      }

 /*
fprintf(stream, "%4d\t\t! Max safe work time [min]\n", max_work);
fprintf(stream, "%4d\t\t! Work-rest cycle [min]\n", work_rest);
fprintf(stream, "%5.2f\t\t! Water rations [canteens]\n", max_water);
fprintf(stream, "%5.1f\t\t! Heat strain casualty [%%]\n", heat_cas);
fprintf(stream, "%5.2f\t\t! Dewpoint [C]\n", measurements.dewpoint);
fprintf(stream, "%5.2f\t\t! Asp_wet_bulb [C]\n",measurements.awet_bulb);
fprintf(stream, "%5.2f\t\t! Nat_wet_bulb [C]\n", measurements.wet_bulb);
fprintf(stream, "%5.2f\t\t! Globe [C]\n", measurements.globe);
fprintf(stream, "%5.2f\t\t! WBGT index [C]\n", measurements.wbgt);
*/
     fprintf(stream, "MxWk[min]\t MxWtr [qt/hr]\t W/R [min]\t CyWtr
     [qt/hr]\t RstWtr [qt/hr]\t RstWtrSh [qt/hr]\tHeat Cas [%%]\t Tdew
     [C]\t Twba[C]\t Twbn[C]\t Tg[C]\t WBGT[C]\t MWkRec[min]\t
     MWkRecS[min]\t patm\n");
     fprintf(stream,"%4d\t%5.1f\t%4d\t%5.1f\t%5.1f\t%5.1f\t%5.1f\t%5.1f\
     t%5.1f\t%5.1f\ t%5.1f\t%5.1f\t%5.0f\t%5.0f\t%5.2f\n", max_work,
     max_water, work_rest, water_req, rest_water, rest_water_sh,
     heat_cas, measurements. dewpoint, measurements.awet_bulb,
     measurements.wet_bulb, measurements. globe, measurements.wbgt,
     maxwk_rcy, maxwk_rcy_sh, patm);

      fclose(stream);
      exit(0);
```

```
}

#define min(x,y) ((x < y) ? x : y)  /* Return the minimum of 2 values */
#define max(x,y) ((x > y) ? x : y)  /* Return the maximum of 2 values */

static float xdata Rload;                    /* Solar factor */
static float xdata TmMWK_41;          /* Max work for light casualties */


/*************************************************************************
*****
 * Function Name:
 *    HsPrediction
 *
 * Description:
 *    This is the main HS prediction routine. It initializes required
 *    variables and calls the routines which calculate work/rest and
 *    water requirements.
 *
 * Inputs:
 *    none
 *
 * Outputs: SUCCESS, if successful
 *          FAILURE, if not
 *

*************************************************************************
****/

int HSPrediction ()
{
HSMALGO_STRUCT xdata hsm;

      ComputeAspiratedWetBulb(&measurements.awet_bulb);

      ComputeDewPoint();

      if (ComputeNaturalWetBulb() == FAILURE)
        return( FAILURE );

      if (ComputeGlobe() == FAILURE)
        return( FAILURE );

      GetSkinTemp (&hsm);

      CalcBaseValues (&hsm);

      DoTheRest (&hsm);

      return( SUCCESS );

} /* end of HSPrediction routine */


/*************************************************************************
*****
 * Function Name:
```

```
 *    DoTheRest
 *
 * Description:
 *
 * Inputs:
 *    none
 *
 * Outputs:
 *    none
 *

*************************************************************************
****/
void DoTheRest (HSMALGO_STRUCT *hsm)
{
float idata temporary;
float idata TmCWKold;
float idata TmCWKnew;
float idata TmMWKest;
float idata TLest;
float MWTL, MCTL;

      /* localize the MAX_WORK_TEMP_LIMIT and MAX_CYCLIC_TEMP_LIMIT
input */
      /* these were hardwired at 39.0 and 38.5 C resp. in orginal hsm_pc
*/
      MWTL = hsm_in.Tmxwork;
      MCTL = hsm_in.Tmxcyc;

 /*************************************************************
 ******************** CEW                        ********************
 *************************************************************/
/*
 * Coefficients for Time Series Equations.
 */
      /* time delay period for work period. */
      hsm->TDWK = 3480. / hsm_in.workrate;

      /*********************************************
         * only for initializing the first work period!
       *********************************************/
      hsm->TreoCWK = 37. + (hsm->TrefRY + hsm->TrefARY - 37.) *
              (pow (0.1, pow (0.4,((hsm->TDWK-30.)/60.))));

      /* time constant for the work period. */
      /* v0.6 change: 120 -> 225: change back to 120, May 2001-wtm ) */
      hsm->KWK = (1. + 3. * exp (0.3 * (hsm->TreoCWK - hsm->TrefWK))) /
120.;

      /* rectal temperature difference for work period. */
      hsm->DTreWK = hsm->TrefWK + hsm->TrefAWK - hsm->TreoCWK;

 /*************************************************************
 ******************** CER                        ********************
 *************************************************************/
```

```
      /* compute time delay period for recovery period */
      if (hsm->CP < 0.)
            hsm->TDRY = 15.;
      else
            hsm->TDRY = 15. * exp (-0.5 * hsm->CP);

      /* time constant for recovery period */
      hsm->KRY = (1. - exp (-1.5 * fabs (hsm->CP)))/40.;


 /*************************************************************
 ******************** MWK                        ***************
 *************************************************************/

      TmMWK_41 = MxWK (MWTL, hsm->DTreWK, hsm->TreoCWK, hsm->TDWK,
                              hsm->KWK);

 /*************************************************************
 ******************** CY                         ***************
 *************************************************************/

/*
 * Compute the work/rest cycle.
 */
      hsm->TmCWK = 0.0;
      hsm->TmCRY = 0.0;
      if (hsm->CP > 0.) {
            TmCWKold = 0.;
            TmCWKnew = 0.;
            if (MCTL > hsm->TrefWK + hsm->TrefAWK)
                  /* Initial rectal temp. for current recovery period */
                  hsm->TreoCRY = hsm->TrefWK + hsm->TrefAWK;
            else
                  hsm->TreoCRY = MCTL;

            /* rectal temperature difference for recovery period */
            hsm->DTreRY = hsm->TrefRY + hsm->TrefARY - hsm->TreoCRY;

            do {
                  /* temporary = RECOVERY_TIME in MERCURY ADA - jrf */
                  temporary = hsm->TDRY + hsm->TDWK + 60. - 0.5 *
                  (TmCWKold + TmCWKnew);

                  if (hsm->TDRY + 5. < temporary)
                        tmtry = temporary;  /* !! THIS IS TIME - jrf */
                  else
                        /* initial rectal tmp for current work period */
                        tmtry = hsm->TDRY + 5.; /* !THIS IS TIME- jrf */

                  /* NOTE: hsm->TreoCWK has units of time on the RHS and
                           temp on the LHS - jrf */
                  hsm->TreoCWK = TreRY (tmtry, hsm->TreoCRY,
                  hsm->DTreRY, hsm->KRY, hsm->TDRY);

                  hsm->DTreWK = hsm->TrefWK + hsm->TrefAWK - hsm-
                  >TreoCWK;
```

```
                TmMWKest = MxWK (MCTL, hsm->DTreWK, hsm->TreoCWK, hsm-
                >TDWK, hsm->KWK);

                TLest = TreWK((TmMWKest + hsm->TDRY), hsm->TreoCWK,
                                hsm->DTreWK, hsm->KWK, hsm->TDWK);

                TmCWKold = TmCWKnew;
                TmCWKnew = MxWK ((MCTL - 0.5 * (TLest - MCTL)),
                                hsm->DTreWK, hsm->TreoCWK,
                                    hsm->TDWK, hsm->KWK);

          } while ( fabs (TmCWKold-TmCWKnew) >= 1.0 );
                /*( round (TmCWKold, 0) != round (TmCWKnew, 0))*/
          hsm->TmCWK = round (TmCWKold, 0);
          if (10. > 60. - hsm->TmCWK)
                hsm->TmCRY = round (hsm->TDRY + 5., 0);
          else
                hsm->TmCRY = 60. - hsm->TmCWK;

     } /* end if CP > 0 */

/*************************************************************
******************** DSP                          ************
*************************************************************/

       TmMWKest = TmMWK_41;
       if (TmMWKest > 300.0) TmMWKest = 300.0;

       if (TmMWKest >= 300.0) {
         hsm->TmCWK = hsm->TmCRY = 0;
         work_rest = 60;
         max_work = 300;
       } else if (hsm->TmCWK < 5) {
         hsm->TmCWK = hsm->TmCRY = 0;
         work_rest = 0;
         if (TmMWKest > 300.0)
            max_work = 300;
         else
            max_work = (int) (round (TmMWKest, 0));
       } else {
         if (hsm->TmCWK >= 300.0) {
            hsm->TmCWK = hsm->TmCRY = 0;
            work_rest = 60;
            max_work = 300;
         } else {
            work_rest = (int) (round (hsm->TmCWK, 0));
         }
         if (TmMWKest > 300.)
            max_work = 300;
         else
            max_work = (int) (round (TmMWKest, 0));
       }

       water_req = max_water = 0.0;
       max_water = (round (hsm->WtrWK, 1));
```

```
            rest_water = (round (hsm->WtrRY, 1));

            rest_water_sh = (round (hsm->WtrRYSh, 1));

        if (hsm->TmCWK + hsm->TmCRY != 0)
              water_req = round ((hsm->TmCWK * hsm->WtrWK +
            hsm->TmCRY * hsm->WtrRY) / (hsm->TmCWK + hsm->TmCRY), 1);


} /* end of main */


/*****************************************************************************
 * Function Name:
 *    CalcBaseValues
 *
 * Notes:
 * Modified 4/13/98 jrf
 * 1) added solider area variable (Asold) and passed to the three
 *    functions requiring it: Hrc, Emax, and WTR.

*****************************************************************************/
void CalcBaseValues (HSMALGO_STRUCT *hsm)
{
#define ALPHA 0.95F           /* soldier absorptivity */

float idata Pa;         /* Ambient air vapor pressure */
float idata HrcWK;
float idata HrcRY;
float idata EreqWK;
float idata EreqRY;

float idata EreqRYSh; /* NEW wtm Aug 01*/
float idata EmaxWK;
float idata EmaxRY;
float idata temp_wk;
float idata temp_ry;
float idata metab_w;
float   Asold, RloadSold,wt,Tmxwork,ereq;

    Asold = DUBOIS(hsm_in.ht, hsm_in.wt);

      wt = hsm_in.wt;

      Tmxwork = hsm_in.Tmxwork;
      RloadSold = Asold/1.8 * Rload;            /* Watts, (Rload is
global for 1.8 m2 man) */

      /* measurements.Lcor = LEWIS_COR(patm);*/

/*
 * Humidity calculations.
 * Pa=ambient air vapor pressure in mmHg (torr)
 */
      Pa = measurements.relhum*SVP(measurements.dry_bulb)/100.;
/*
 * Get the metabolic rate in watts for the work period.
```

```
 * (WTM disabled hard categories:resting = 105  very light = 150  light
= 250  moderate = 425  heavy = 600)
 */

      metab_w = hsm_in.workrate;
/*
 * Clothing related calculations.
 */
      temp_wk = Veff(metab_w);
      temp_ry = Veff(105.);

      /* convective and radative exchange for   work period. */
      /* It(temp_wk) is clothing insulation value for work period */
      HrcWK = Hrc(It(patm, temp_wk), hsm, Asold);

      /* convective and radative exchange for recovery period. */
      /* It(temp_ry) is clothing insulation value for recovery period */
      HrcRY = Hrc(It(patm, temp_ry), hsm, Asold);

      /* maximum evaporative loss for work period. */
      /* Cevap(temp_wk) is clothing im/clo value for work period */
      EmaxWK = Emax(patm, Cevap(temp_wk), Pa, hsm, Asold);

      /* maximum evaporative loss for recovery period. */
      /* Cevap(temp_ry) is clothing im/clo value for recovery period */
      EmaxRY = Emax(patm, Cevap(temp_ry), Pa, hsm, Asold);

      temp_wk = U(temp_wk);    /* clothing efficiency factor for work
                                  period */
      temp_ry = U(temp_ry);    /* clothing efficiency factor for
                                  recovery period */

      /* required evaporative loss for work period. */
      EreqWK = Ereq(HrcWK, metab_w, temp_wk, RloadSold);
      /* required evaporative loss for recovery period. */
      EreqRY = Ereq(HrcRY, 105., temp_ry, RloadSold);
      /* NEW: required evaporative loss for rcovery in shade */

      EreqRYSh = Ereq(HrcRY, 105., temp_ry, 35.);

      /* NEW: recovery time from max work in the sun */

      maxwk_rcy = MxWRKRY (wt,Tmxwork, EmaxRY, EreqRY);

      /* NEW: recovery time from max work in the shade*/

      maxwk_rcy_sh = MxWRKRY (wt,Tmxwork, EmaxRY, EreqRYSh);
      hsm->CP = 0.015 * (EmaxRY - EreqRY);       /* Cooling power */

      hsm->CPSh = 0.015 * (EmaxRY - EreqRYSh);  /* Cooling power in
      shade */
/*
 * Final Rectal and Delta Final Rectal for Acclimatization.
 */
      /* Final rectal temperature for work period. */
      hsm->TrefWK = Tref(metab_w, temp_wk, HrcWK, EreqWK, EmaxWK,
      RloadSold);
```

```
      /* Final rectal temperature for recovery period. */
      hsm->TrefRY = Tref(105., temp_ry, HrcRY, EreqRY, EmaxRY,
      RloadSold);
      /* Final rectal temperature add-on for work period. */
      hsm->TrefAWK = DTref(hsm->TrefWK, EmaxWK);
      /* Final rectal temperature add-on for recovery period. */
      hsm->TrefARY = DTref(hsm->TrefRY, EmaxRY);
/*
 * Water Requirements.
 */
      /* water requirements in qt/hr for work period. */
      hsm->WtrWK = WTR (EreqWK, EmaxWK, Asold);
      /* water requirements in qt/hr for recovery period. */
      hsm->WtrRY = WTR (EreqRY, EmaxRY, Asold);
      /* water requirement in quarts for recovery period in the shade */

      hsm->WtrRYSh = WTR (EreqRYSh,EmaxRY, Asold);
      /*  compute heat casualty */
    heat_cas = heatcas(hsm->TrefWK + hsm->TrefAWK);

} /* end of calc_base_values */



/*************************************************************************
*****
 * Function Name:
 *    ComputeAspiratedWetBulb
 *
 * Description: Computes wet bulb from dry bulb and RH.
 *
 * Notes:
 * 1) the factor 2.0 is the Lewis number.  It has units of degrees C /
mm Hg

 * 2) Lewis number correction for atmospheric pressure Lcor= Patm^(-.45)

*************************************************************************
****/
void ComputeAspiratedWetBulb (float *Twb)
{
float idata k1;
float idata k2;
float idata delta;
float idata awet_bulb;

   k1 = measurements.Lcor * 2.0 * (SVP (measurements.dry_bulb) *
measurements.relhum / 100.0) + measurements.dry_bulb;
   k2 = 0.0;
   delta = 10.0;
   awet_bulb = 0.;

   while (1) {
      while (k1 - k2 >= 0.) {
         k2 = measurements.Lcor * 2.0 * SVP (awet_bulb) + awet_bulb;
         awet_bulb += delta;
```

```
            if (awet_bulb > 100.0) break;
        }
        awet_bulb -= measurements.Lcor * 2.0 * delta;
        delta /= 10.0;
        if (delta < 0.0001) break;
        awet_bulb -= delta;
        k2 = 0.0;
    }
    measurements.awet_bulb = awet_bulb;
}


void ComputeDewPoint()
/*********************************************************************
  computes the dewpoint from the dry bulb temperature and relative
humidity

*********************************************************************/
{
float Pdp, logPdp;

      Pdp = 0.01 * measurements.relhum * SVP(measurements.dry_bulb);
      /* mm Hg */
      logPdp = (float)log10(Pdp);
      measurements.dewpoint = (155.0F - 235.0F * logPdp) / (logPdp -
8.1076F);
}



int ComputeNaturalWetBulb()
/*********************************************************************
  computes the natural wet bulb temperature using the model of Gonzalez
et al.

  Computes by solving the heat balance

                              f = Qin(Twb) - Qout(Twb) = 0

  using the secant iterative method

                              x_n+1 = x_n - f(x_n) / m
                              m = (f(x_n) - f(x_n-1)) / (x_n - x_n-1).

  where,

      Qin(Twb) = Qconv + Qrad
      Qout(Twb) = Qevap

*********************************************************************/
{
#define MAXITER2 10

int ict;
float Ta, Trk, dp, hconv, hevap, Trk4, Pdp;
float xnew, xold, fnew, fold, m, reldif, scale;
```

```
    Ta = measurements.dry_bulb;
      Trk = measurements.mrt + 273.16F;
      dp = measurements.dewpoint;

      hconv = 42.024F * pow((patm*measurements.wind_speed), 0.466);
      hevap = measurements.Lcor * 2.2F * hconv;
      Trk4 = Trk * Trk * Trk * Trk;
      Pdp = SVP( dp );                          /* mm Hg */

/*********************************************************************

    Let our first guess for wet bulb be equal to the dewpoint.
    For semantic purposes we name xnew first and xold
    second to land the first iteration on (xold, fold).


*********************************************************************/
      xnew = dp;
      fnew = f_wetbulb(hconv, hevap, Ta, Trk4, Pdp, xnew);
      xold = xnew * 1.001;
      fold = f_wetbulb(hconv, hevap, Ta, Trk4, Pdp, xold);
      m = (fold - fnew) / (xold - xnew);

/*  need to define "zero" */
      scale = fabs((xnew + 273.16F) * 1.e-6);

      ict = 0;

lin_10:
    if (ict > MAXITER2)
        return( FAILURE );

      xnew = xold - fold / m;
      reldif = fabs(xnew - xold);
      if (reldif > scale) {
        fnew = f_wetbulb(hconv, hevap, Ta, Trk4, Pdp, xnew);
        m = (fnew - fold) / (xnew - xold);
        fold = fnew;
        xold = xnew;
        ict = ict + 1;
        goto lin_10;
        }

      measurements.wet_bulb = xnew;
      return( SUCCESS );
}

/*********************************************************************

   f_wetbulb

 *********************************************************************/

float f_wetbulb(float hconv, float hevap, float Ta, float Trk4,
                      float Pdp, float Twb)
/*********************************************************************
```

```
 the heat balance equation used to compute the natural wet bulb
temperature

 WTM changes absorbtivity/emissicvity of wick from 1 to 0.4* STEFAN_B

*****************************************************************/
{
float Twbk, Twbk4, Pwb, Qconv, Qevap;

      Twbk = Twb + 273.16F;
      Twbk4 = Twbk * Twbk * Twbk * Twbk;
      Pwb = SVP( Twb );
      Qconv = hconv * (Ta - Twb) + 0.4*STEFAN_BOLTZMANN * (Trk4 -
Twbk4);
      Qevap = hevap * (Pwb - Pdp);
      return(Qconv - Qevap);
}



/*****************************************************************
 * Function Name:
 *    ComputeGlobe
 *
 * Computes the globe temperature for a 6 inch globe given a mean
 * radiant temperature and the properties of the air.
 *
 * Also computes Rload (a global variable), the radiant load [W/m2]
 * on the solider, and the wbgt index.
 *
 * Outputs: SUCCESS, if successful
 *          FAILURE, if not
 *
 *****************************************************************/
int ComputeGlobe (void)
{


float d;                                          /* diameter of
globe */
float v;                                          /* velocity of air
*/
float xdata Tr;                                   /* mean radiant
temperature */
float xdata Tgt;                                /* globe temperature */
float xdata Ta;                                   /* ambient
temperature */
float xdata Hcg;                                  /* convective heat
exchange coef */
float xdata Hrg;                                  /* radiative heat
exchange coef */
float Ta2, Tr2;

    Ta = measurements.dry_bulb + 273.16;

      Ta2 = Ta * Ta;
```

```
      Tr = measurements.mrt + 273.16;
      Tr2 = Tr * Tr;
    d = 0.1524;         /* meters, for 6" globe */
    v = measurements.wind_speed; /* m/s */
      Hrg = 0.95 * STEFAN_BOLTZMANN * (Tr + Ta) * (Tr2 + Ta2);
    Hcg = 6.32 * pow (d, -0.4) * pow ((patm * v), 0.5);           /*
patm is bar pres correction */

      Tgt = (Hcg * Ta + Hrg * Tr)/(Hcg + Hrg);


   measurements.globe = Tgt;
   measurements.globe -= 273.16;

/*
 * Determine the radiant load [W] using USARIEM Technical Report
No.T01/13
 */


   Rload = -0.071 * pow ((Tr - Ta), 2) + 10.432 * (Tr - Ta); /* W */

/*
 * Determine the WBGT.
 */
   measurements.wbgt = 0.7 * measurements.wet_bulb + /* outdoor expr */
                             0.2 * measurements.globe +
                             0.1 * measurements.dry_bulb;

   return( SUCCESS );
}


/**************************************************************************
*****
 * Function Name:
 *    SVP
 *

**************************************************************************
****/
float SVP (float T)
{
      return (pow (10., (8.1076-(1750.286/(T+235.)))));

} /* end of SVP routine */


/**************************************************************************
*****
 * Function Name:
 *    Veff
 *

**************************************************************************
****/
float Veff (float M)
```

```
{
      return ((measurements.wind_speed)+0.004*(M-105.));

} /* end of Veff routine */


/*************************************************************************
* Function Name:
*     It
*
*************************************************************************/
float It(float patm, float V)
{
float Itc;
float Itvc;
clothing_table *cl_table;

/*
 * Set default clothing coefficients.
 */
        cl_table =
clothing_list[input_selections.clothing_type_idx].clo_lst;

      /* clothing coefficient */
        Itc  = cl_table[input_selections.clothing_idx].itc;

      /* clothing velocity coefficient */
        Itvc = cl_table[input_selections.clothing_idx].itvc;

      return (Itc * pow((patm * V), Itvc));

} /* end of It routine */


/*************************************************************************
 * Function Name:
 *    Cevap
 *
*************************************************************************/
float Cevap (float V)
{
float Imc;
float Imvc;
clothing_table *cl_table;

/*
 * Set default clothing coefficients.
 */
        cl_table =
clothing_list[input_selections.clothing_type_idx].clo_lst;


      /* clothing permeability coefficient */
        Imc  = cl_table[input_selections.clothing_idx].imc;

      /* clothing permeability velocity coefficient */
        Imvc = cl_table[input_selections.clothing_idx].imvc;
```

```
      return (Imc*pow(V, Imvc));

} /* end of Cevap routine */


/*************************************************************************
 * Function Name:
 *    U
 *

*************************************************************************/
float U (float V)
{
float Itc;
float Itvc;
clothing_table *cl_table;

/*
 * Set default clothing coefficients.
 */
        cl_table =
clothing_list[input_selections.clothing_type_idx].clo_lst;


      /* clothing coefficient */
        Itc  = cl_table[input_selections.clothing_idx].itc;

      /* clothing velocity coefficient */
        Itvc = cl_table[input_selections.clothing_idx].itvc;

      return ((0.41/Itc)*pow(V, (-(0.43+Itvc))));

} /* end of U routine */


/*************************************************************************
 * Function Name:
 *    Hrc
 *
 * Notes:
 * Modified 4/13/98 jrf
 * 1) the original constant "11.772" has been replaced by the product
 *    6.45 * A
 *    where
 *      A is the DuBois surface area
 *    This is consistent with the Ada implementation of HSDAN.

*************************************************************************/
float Hrc (float It, HSMALGO_STRUCT *hsm, float A)
{

      return (6.45F * A *(measurements.dry_bulb - hsm->Tskin)/It);

} /* end of Hrc routine */
```

```
/*****************************************************************
 * Function Name:
 *    Ereq
 *

*****************************************************************/
float Ereq (float Hrc, float M, float U, float RloadSold)
{
      return (Hrc + M + U * RloadSold);

} /* end of Ereq routine */


/*****************************************************************
 * Function Name:
 *    Emax
 *
 * Notes:
 * Modified 4/13/98 jrf
 * 1) the original constant "25.935" has been replaced by the product
 *    14.21 * A
 *    where
 *      A is the DuBois surface area
 *    This is consistent with the Ada implementation of HSDAN.
 * 2) the constant 45.8 has been replaced by the function SVP, given the
 *     skin temperature.
 * 3) Emax factor for high terrestrial altitude, patm ^-0.45  wtm Nov
2001
*****************************************************************/
float Emax (float patm, float Cevap, float Pa, HSMALGO_STRUCT *hsm,
float A)
{
float result;


    result = (pow(patm, -0.45) * 14.21F * A * Cevap * (SVP (hsm->Tskin)
- Pa));


/*    result = (14.21F * A * Cevap * (SVP (hsm->Tskin) - Pa));*/
      return( result );

/* 4/13/98 jrf return (25.935 * Cevap * (45.8 - Pa)); */

/* 3/19/97 KSH return (25.935 * Cevap * (SVP (hsm->Tskin) - Pa)); */

} /* end of Emax routine */

 /*****************************************************************
 * Function Name:
 *    Tref
 *

*****************************************************************/
float Tref (float M, float U, float Hrc, float Ereq, float Emax,
                  float RloadSold)
```

```
{
      return (36.75 + 0.004*M + 0.0025* U * RloadSold + 0.0011*Hrc +
            0.8*exp(0.0047*(Ereq - Emax)));

} /* end of Tref routine */


/*************************************************************************
 * Function Name:
 *    DTref
 *
 * Notes:
 *
 * Modified 4/13/98 by jrf
 * 1) added logic for Kdehyd
 * 2) acclimation assigned directly to days in heat (DIH)
 * 3) broke big eq into smaller, more readable pieces
 * 4) removed
 *    if (DTref_tmpry < 0.0001F)
 *      DTref_tmpry = 0.0F;
 *    logic

*************************************************************************/
float DTref (float Tref, float Emax)
{
float DTref_tmpry;
float DIH;
float Kdehyd, alp;

      /* days in heat */
    DIH = hsm_in.acclim;

      /* dehydration factor */
      if (hsm_in.dehyd >= 5.0F)
        Kdehyd = 1.0F;
      else
        Kdehyd = 0.2F * hsm_in.dehyd;

      alp = -0.3F * DIH * (1 - Kdehyd);

      DTref_tmpry = 0.0;
      if (Emax > 0.) {
            DTref_tmpry = (0.5F +
                              1.2F *
                                  (1.0F - exp(0.5*(37.15 - Tref))) *
                        (1.0F - exp(-0.005*Emax)) );
            DTref_tmpry *= (float)exp(alp);
            DTref_tmpry += 0.1735F * hsm_in.dehyd - 0.215F;

/*          if (DTref_tmpry < 0.0001F)
                  DTref_tmpry = 0.0F; */
      }

      return (DTref_tmpry);

} /* end of DTref routine */
```

```
/*****************************************************************
 * Function Name:
 *    WTR
 *
 * Notes:
 * Modified 4/13/98 jrf
 * 1) the original constant "50.921" has been replaced by the product
 *    27.9 * A

 * Modified March 2001 wtm

 * 1) original Shapiro Ereq and Emax are per unit BSA!
 *    where
 *      A is the DuBois surface area
 *    This is consistent with the Ada implementation of HSDAN.

*****************************************************************/
float WTR (float Ereq, float Emax, float A)
{
float sweat;

/* new variables, body surface area-normalized values of Ereq and
Emaxfor Shapiro Eq. wtm */

float Ereqs;

float Emaxs;

   Ereqs=Ereq/A;

   Emaxs = Emax/A;

      if (Emaxs <= 0.0)
            sweat = 2000.;
      else
            sweat = 27.9F * A * Ereqs * pow(Emaxs, -0.455);
/*
 * Limit sweat to the range 150 - 2000.
 */
      sweat = max ((min (sweat, 2000.)), 150.);
/*
 * Return the water requirements.
 */
      return (sweat * .0010567);    /* liters -> canteens (i.e. qts) */

} /* end of WTR routine */


/*****************************************************************
 * Function Name:
 *    MxWK
 *

*****************************************************************/
float MxWK (float TL, float DTreWK, float TreoCWK, float TDWK, float
KWK)
```

```
{
float temp;

      if (DTreWK > 0.) {
            temp = (DTreWK + TreoCWK - TL) / DTreWK;
            if (temp > 0.)
                  temp = log (temp);
            else
                  temp = -227.9; /* same as log (1e-99) */

      } else {
            temp = -227.9; /* same as log (1e-99) */
      }
      return (TDWK - temp/KWK);

} /* end of MxWK routine */



/*****************************************************************
 * Function Name:
 *    round
 *
*****************************************************************/
float round (float x, unsigned char digit)
{
      x *= pow (10.0, (float)digit);
   x = (float)((int)(x + 0.5));
      x /= pow (10.0, (float)digit);

      return (x);

} /* end of round routine */


/*****************************************************************
 * Function Name:
 *    TreWK
 *
*****************************************************************/
float TreWK (float t, float TreoCWK, float DTreWK, float KWK, float
TDWK)
{
      return (TreoCWK + DTreWK * (1.0 - exp (-KWK * (t - TDWK))));

} /* end of TreWK routine */


/*****************************************************************
 * Function Name:
 *    TreRY
 *
*****************************************************************/
```

```
float TreRY (float t, float TreoCRY, float DTreRY, float KRY, float
TDRY)
{
      return (TreoCRY + DTreRY * (1.0 - exp (-KRY * (t - TDRY))));

} /* end of TreRY routine */



/*************************************************************************
 * Function Name:
 *    MxWKRY
 *
 *    note:This is Lee's StrTm,cooling based on heat content,
 *  0.965 is 0.83 body sp heat / 0.8606 Kcal/hr/Watt
 *  wtm Aug 2001
 *
 *************************************************************************/
float MxWRKRY (float wt, float Tmxwork, float EmaxRY, float ereq)

{

float temp;

      temp = (0.95 * wt * (Tmxwork - 37.0)/(EmaxRY - ereq))*60;

      if (temp < 0 ) temp = 0;

      return (round(temp, 1));


} /* end of MxWRKRY


/*************************************************************************
 * Function Name:
 *    GetSkinTemp
 *
 *    note: dead code
 *   Hold Tskin to 36.5
*************************************************************************/
void  GetSkinTemp (HSMALGO_STRUCT *hsm)
{
#if 0
   float T0;
   float Hc;

   Hc = 8.6 * pow (measurements.wind_speed, 0.5);

   T0 = (4.7 * MRT + Hc * measurements.dry_bulb) / (4.7 + Hc);

   hsm->Tskin = 25.8 + 0.267 * T0;
/*
 * Constrain to 34 - 37.
 */
   hsm->Tskin = min (37.0, max (34.0, hsm->Tskin));
```

```
#else

   hsm->Tskin = 36.5;
#endif

} /* end of GetSkinTemp routine */
```